

10/24/00



JC957 U.S. PTO

10-26-00

A

JC929 U.S. PTO
09/695813

10/24/00

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventorship Levi et al.
Applicant Microsoft Corporation
Attorney's Docket No. MS1-626US
Title: System and Method for Logical Modeling of Distributed Computer Systems

TRANSMITTAL LETTER AND CERTIFICATE OF MAILING

To: Commissioner of Patents and Trademarks,
Washington, D.C. 20231

From: Brian G Hart (Tel. 509-324-9256; Fax 509-323-8979)
Lee & Hayes, PLLC
421 W. Riverside Avenue, Suite 500
Spokane, WA 99201

The following enumerated items accompany this transmittal letter and are being submitted for the matter identified in the above caption.


1. Specification--title page, plus 30 pages, including 32 claims and Abstract
2. Transmittal letter including Certificate of Express Mailing
3. 5 Sheets Formal Drawings (Figs. 1-6)
4. Return Post Card

Large Entity Status ☒

Small Entity Status ☐

Date: 10-24-2000

By:


Brian G Hart
Reg. No. 44,421

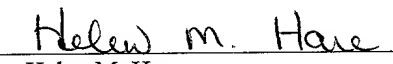
CERTIFICATE OF MAILING

I hereby certify that the items listed above as enclosed are being deposited with the U.S. Postal Service as either first class mail, or Express Mail if the blank for Express Mail No. is completed below, in an envelope addressed to The Commissioner of Patents and Trademarks, Washington, D.C. 20231, on the below-indicated date. Any Express Mail No. has also been marked on the listed items.

Express Mail No. (if applicable) EL685271192

Date: Oct. 24, 2000

By:


Helen M. Hare

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**System and Method for Logical Modeling of Distributed
Computer Systems**

Inventor(s):

Steven Levi

Galen Hunt

Aamer Hydrie

Jakob Rehof

Bassam Tabbara

Mark VanAntwerp

Robert Welland

TECHNICAL FIELD

This invention relates to distributed computer systems, such as Websites and Internet-based Services. More particularly, this invention pertains to a way to model an application for a distributed computer system in a scale-invariant manner that is abstracted from the underlying physical configuration.

BACKGROUND

It is no secret that Internet usage has exploded over the past few years and continues to grow rapidly. People have become very comfortable with many services offered on the World Wide Web (or simply "Web"), such as electronic mail, online shopping, gathering news and information, listening to music, viewing video clips, looking for jobs, and so forth. To keep pace with the growing demand for Internet-based services, there has been tremendous growth in the computer systems dedicated to hosting Websites, providing backend services for those sites, and storing data associated with the sites.

One type of distributed computer system is an Internet data center (IDC), which is a specifically designed complex that houses many computers used for hosting Internet-based services. IDCs, which also go by the names "webfarms" and "server farms", typically house hundreds to thousands of computers in climate-controlled, physically secure buildings. These computers are interconnected to run one or more programs supporting one or more Internet services or Websites. IDCs provide reliable Internet access, reliable power supplies, and a secure operating environment.

Fig. 1 shows an Internet data center 100. It has many server computers 102 arranged in a specially constructed room. The computers are general-purpose

1 computers, typically configured as servers. An Internet data center may be
2 constructed to house a single site for a single entity (e.g., a data center for Yahoo!
3 or MSN), or to accommodate multiple sites for multiple entities (e.g., an Exodus
4 center that host sites for multiple companies).

5 The IDC 100 is illustrated with three entities that share the computer
6 resources: entity A, entity B, and entity C. These entities represent various
7 companies that want a presence on the Web. The IDC 100 has a pool of additional
8 computers 104 that may be used by the entities at times of heavy traffic. For
9 example, an entity engaged in online retailing may experience significantly more
10 demand during the Christmas season. The additional computers give the IDC
11 flexibility to meet this demand.

12 While there are often many computers, an Internet service or Website may
13 only run a few programs. For instance, one Website may have 2000-3000
14 computers but only 10-20 distinct software components. Computers may be added
15 daily to provide scalability as the Website receives increasingly more visitors, but
16 the underlying programs change less frequently. Rather, there are simply more
17 computers running the same software in parallel to accommodate the increased
18 volume of visitors.

19 Today, there is no conventional way to architect Internet Services in a way
20 that abstracts the functionality of the Service from the underlying physical
21 implementation. Little thought has gone into how to describe a complete Internet
22 Service in any manner, let alone a scale-invariant manner. At best, Internet
23 Service operators might draft a document that essentially shows each and every
24 computer, software program, storage device, and communication link in the center
25 as of a specific time and date. The downside with such physical schematics is, of

1 course, that the document is always out of date and hence, it is of limited
2 usefulness as a management tool. Furthermore, while a human may understand
3 such a document, it holds no meaning to a computer.

4 5 **SUMMARY**

6 A modeling system permits developers to architect distributed computer
7 applications, such as Internet Services or Websites, in an abstract manner. The
8 modeling system defines a set of components that represent functional units of the
9 applications that will eventually be physically implemented by one or more
10 computers and one or more software programs executing on the computers.

11 In the described implementation, the modeling system defines several
12 model components: a module, a port, and a wire; and a set of model extensions
13 including, but not limited to: a store, an event source, an event sink, and an event
14 wire.

15 The module is the basic functional unit and represents a container of
16 behavior that may be implemented by one or more computers running one or more
17 software programs. For instance, in the context of a Website, one module might
18 represent a front end that renders HTML pages, another module might represent a
19 login database, and another module might represent a mailbox program. A port is
20 a service access point for the module. All communications into and out of the
21 module goes through a port. A wire is the logical binding that defines an allowed
22 communication route between two ports.

23 While the model consists of the three basic components described above
24 (namely modules, ports, and wires), the model can be augmented with numerous
25 extensions, specializations of the basic components. For example, a store is a

1 basic unit of storage and a specialization of the module. A store represents a
2 logical amount of storage, which may be implemented by any number of physical
3 disks or other storage media. Like the module, the store represents behavior, in
4 the case the ability to save and retrieve data. Also like the module, the store can
5 communicate with other modules and stores through ports and wires. A store
6 differs from a module in that it is labeled with additional attributes such as the
7 amount of storage required, required access speed, or a minimum number of
8 outstanding storage requests. The store extends the model by adding a specialized
9 type of module with additional semantic information.

10 The model can be further augmented with ports extensions. For example,
11 an event source and an event sink are used for discrete semantic messaging
12 between modules and module extensions, such as stores. Event sinks are
13 specialized ports in that they are communication access points between model
14 components, but with additional semantics, namely the specific events.

15 The model can also be augmented with wires extensions. For example, an
16 event wire is a logical connection between event sources and event sinks, and
17 carries event messages used to inform modules and implement policy. While most
18 wire extensions allow communication at run time, it is possible for some wire
19 extensions to transfer data only at compile or initialization time.

20 The model components are arranged and interconnected to form a scale-
21 independent model of the application. Each component specifies some
22 functionality of the application.

23 Associated with the model components is a schema that dictates how the
24 functional operations represented by the components are to be specified. For
25 example, the schema might dictate that a module specify processing requirements,

1 software programs needed to implement the behavior, other modules with which
2 the module should communicate, and so forth. The schema might further dictate
3 that a port and a wire specify a set of attributes describing format, semantics,
4 protocol, and so on. The schema might further dictate that extensions of modules,
5 port, and wires specify further attributes.

6 From the model components, the developers can create logical, scale-
7 independent models of the applications that may be implemented by the
8 distributed computer systems. An application is scale-independent in that it is
9 invariant with respect to the number of computers and software programs that may
10 eventually be used to implement it. The application may subsequently be
11 converted to a physical blueprint that specifies the number and type of
12 hardware/software resources and the physical layout of the distributed computer
13 system.

14 **BRIEF DESCRIPTION OF THE DRAWINGS**

15 Fig. 1 illustrates a conventional Internet data center (IDC).
16

17 Fig. 2 illustrates a set of model components that form the building blocks
18 for modeling an Internet Service, along with the associated schema.

19 Fig. 3 illustrates a database application for an Internet Service that is
20 modeled in terms of the components.

21 Fig. 4 illustrates an Internet-based email Internet Service.

22 Fig. 5 is a block diagram of a computer that may be used to implement the
23 modeling software for modeling the Internet Service.

24 Fig. 6 is a flow diagram of a process for modeling an Internet Service.
25

DETAILED DESCRIPTION

A modeling system permits developers of distributed computer applications (e.g., Internet Services, Websites, and the like) to architect their hardware and software in an abstract manner. The modeling system defines a set of components used to describe the functionality of an application in a logical, scale-independent manner. An “application” within this context refers to an entire service hosted on the distributed computers. For instance, an Internet data center may host a Website for an online retailer, where the application entails the entire software and hardware configuration that implements the online retailer’s Internet presence. The application might include, for example, a front end to handle client requests, an order processing system, a billing system, an inventory system, and a database system.

The model components are arranged and interconnected to form a scale-independent model of the application. Each component specifies some functionality of the application. The model can then be used to construct a scalable physical blueprint in terms of which machines run which pieces of software to form the application. The model continues to be helpful for managing ongoing operations of the application, forming the skeleton upon which operational behavior and policy mechanisms are built.

In this manner, the modeling system changes the development effort from a node-centric approach to an application-centric approach. With conventional node-centric methodology, the focus was on the computers and how they were laid out. The application was then implemented on as many nodes as needed. With the new application-centric approach, the focus is initially on the application itself.

1 The physical nodes used to implement the application are derived once the
2 application is created.

3 The modeling system is described in the context of Internet Services and
4 Websites, such as might be deployed in Internet Data Centers, because modeling
5 Internet Services represents one suitable use of the system. However, the
6 modeling system may be implemented to model other large size and scalable
7 computer systems. Accordingly, the modeling system can be implemented in a
8 wide variety ways, including both Internet-based implementations and non-
9 Internet-based implementations.

10 11 **Model Components and Schema**

12 The modeling system defines several model components that form the
13 building blocks of a logical, scale-independent application: a module, a port, and a
14 wire. It also defines a set of model extensions including, but not limited to: a
15 store, an event source, an event sink, and an event wire. In a design tool, the
16 components are represented pictorially as graphical elements or symbols that may
17 be arranged and interconnected to create scale-independent models of Website
18 applications. The graphical elements have an associated schema that dictates how
19 the functional operations being represented by the graphical elements are to be
20 specified.

21 Fig. 2 illustrates a set of model components 200 that form the building
22 blocks of logical, scale-independent Internet Services. The components include a
23 module, as represented by modules 202(A)-202(C), ports 206, wires 208, and
24 extensions such as a store 204, event sources 210, event sinks 212, and event wires
25

1 214. The components 200 are arranged in a no particular manner other than to
2 foster discussion of their individual traits.

3 A module 202 represents a basic unit of functionality for the Internet
4 Service. It is a logical entity that represents some portion of the application as
5 might be deployed at the IDC, but it does not necessarily have a physical
6 manifestation. The module often corresponds to a software program that handles a
7 logical set of tasks for the Service. For instance, one module might represent a
8 front end for a Website, another module might represent a login database, and
9 another module might represent an electronic mail program.

10 Each module 202 is a container of behavior. A simple module is indivisible
11 and has associated a unique identifier. Modules can be nested into a hierarchy of
12 modules to form more complex behaviors. In a module hierarchy, the leaf
13 modules are simple modules, and the non-leaf modules are compound modules.

14 Each module 202 defines a unit of scaling. While one module logically
15 represents a functional operation of the Service, the module may be deployed to
16 any number of computers when actually implemented. In this way, the module is
17 scale-independent, allowing the number of underlying computers used to
18 implement the module to change at over time. When converted to a physical
19 implementation, "module instances" are created from the modules. The module
20 instances are assigned a unique identifier and maintain ancestral data regarding
21 which module created them. The module instances of simple modules are often
22 called "engines", which are software programs that run on individual computers.

23 A port 206 is a service access point (SAP) for a module 202 or store 204.
24 All service-related communications into and out of a module go through a port
25 206. Each port 206 has a "type", which is a set of attributes describing format,

1 semantics, protocol, and so forth. At runtime, the port represents a set of physical
2 ports associated with the instantiated engines of the modules. Note that a given
3 module might have any number of ports representing different services or
4 functionality provided by the module.

5 A wire 208 is the logical binding that defines an allowable communication
6 route between two ports 206. Each wire 208 can be type-checked (i.e., with
7 respect to protocols, roles) and defines protocol configuration constraints (e.g.,
8 HTTP requires TCP, TCP requires IP, etc.).

9 Extensions to the model are additional components that specialize the role,
10 behavior, and possibly graphical representation of the base components.
11 Exemplary extensions include, but are not limited to, store 204, event source 210,
12 event sink 212, and event wire 214.

13 A store 204 is the most basic unit of storage. It represents a logical storage
14 partition, which may be implemented by any number of physical disks or other
15 storage media.

16 Event sources 210 and event sinks 212 are used for discrete semantic
17 messaging between modules and module extensions, such as stores. An event wire
18 214 is a logical connection between sources and sinks, and carries event messages
19 used to inform modules or module extensions and implement policy (e.g., scaling,
20 fail-over, monitoring, application processes, etc.).

21 The event sources 210 and event sinks 212, together with the ports 206,
22 collectively form interfaces for communications to and from the modules 202 and
23 module extensions, such as stores 204. The event sources and sinks may be
24 implemented as ports that are configured for message handling.

1 The model components 200 are depicted as graphical icons or symbols that
2 may be selected and interconnected using a modeling system (described below in
3 more detail). In the illustrated example, the modules 202 are depicted as blocks,
4 the store 204 is depicted as a disk storage icon, and the ports 206 are depicted as
5 spherical knobs projecting from the modules or module extensions, such as stores.
6 Additionally, the wires 208 are depicted as bold lines, the event sources 210 are
7 depicted as triangles pointing away from the module or module extension, the
8 event sinks 212 are depicted as triangles pointing toward the module or module
9 extension, and the event wire 214 is depicted as a dashed line.

10 The graphical icons have an associated schema that dictates how the
11 functional operations being represented by the icons are to be specified. For
12 instance, a module icon may have a predefined schema that specifies the hardware
13 and software resources used to implement the functionality represented by the
14 module. Thus, a module for a database function might have characteristics
15 pertaining to the kind of database (e.g., relational), the data structure (e.g., tables,
16 relationships), software (e.g., SQL), software version, and so forth.

17 Fig. 2 also illustrates the schema underlying the graphical elements as
18 exemplary data structures associated with the model components. Module 202(A)
19 has an associated structure 220 that contains various characteristics for the
20 module, such as functionality, processing requirements, software, and so forth.
21 Modules 202(B) and 202(C) have similar structures (not shown). Model
22 extensions also have associated structures. The store 204 has a corresponding
23 structure 222 that defines the requirements for storage. The store schema structure
24 222 might include, for example, the kind of storage (e.g., disk), the storage format,
25 and so on.

Each port 206 has a schema structure, as represented by structure 224, which dictates the port's type. Each wire 208 is also associated with a schema structure, such as structure 226, which outlines the protocols implemented by the connection. Similar schema structures may also be provide for event sources, event sinks, and event wires.

Using the model components, a developer can logically describe and configure scale-independent Internet Service prior to physically laying them out in Internet data centers. The developer drafts a model using a user interface to select and interconnect the model components. Once constructed, the modeling software generates the Internet Service based on the depicted model and the underlying schema. The Service may subsequently be converted into a physical blueprint that details the computers and software needed to implement the Service for a specified number of clients.

The scale-invariant nature of the modeling system allows Internet Service developers to focus only on designing software for a specific functional task (e.g., front end, login database, email program, etc.). All external communications can then be expressed in terms of transmitting to and receiving from one or more associated ports. In this manner, the Service developers need not worry about how many machines will be used to run the module, or how other modules of the scale-independent Internet Service are being configured.

Exemplary Module and Application

Fig. 3 shows a fault-tolerant SQL (structure query language) database module 300 to demonstrate how the model components may be organized and connected to represent a portion of an application. In this example, the database

1 module 300 represents a SQL database that may be used independently or as a
2 component in a larger application. The SQL database module 300 has a module
3 interface composed of a single port 302 that implements the TDS (Tabular Data
4 Stream) protocol.

5 The SQL database module 300 is a compound module made up of three
6 simple modules: a fail-over policy module 310, a primary SQL module 312, and a
7 secondary SQL module 314. The primary and secondary SQL modules represent
8 dual programs that operate in parallel so that, in the event that the primary module
9 312 crashes, the secondary module 314 can assume the role without loss of
10 service. The database module 300 also has a data store 316 that represents the
11 memory storage for the SQL database module.

12 The primary SQL module 312 has a module interface that includes a first
13 port 320 for communicating with the compound module port 302 and a second
14 port 322 for communicating with the store 316. The primary SQL module 312
15 also has an event source 324 and an event sink 326 for handling event messages
16 from the fail-over policy module 310. Similarly, the secondary SQL module 314
17 has a module interface with a first port 330 for communicating with the compound
18 module port 302, a second port 332 for communicating with the store 316, and an
19 event sink 334 for receiving events from the fail-over policy module 310. A wire
20 336 interconnects the external compound module port 302 with the ports 320 and
21 330 of the primary and secondary SQL modules, respectively.

22 The store 316 has a port 340 to communicate with the primary and
23 secondary SQL modules 312 and an event sink 342 to receive event messages
24 from the fail-over policy module 310. A wire 344 interconnects the store port 340
25

1 with the ports 322 and 332 of the primary and secondary SQL modules,
2 respectively.

3 The fail-over policy module 310 has a module interface that includes three
4 event sources and one event sink. An event sink 350 receives a “fail” event from
5 the event source 324 of the primary SQL module 312 via an event wire 352 when
6 the primary SQL module experiences some failure. In response to receiving a
7 “fail” event, the fail-over policy module 310 concurrently issues a first event to
8 stop the failed primary module 312, another event to assign the secondary module
9 314 as the new owner of the store 316, and a third event to start the secondary
10 module 314. The “stop” event is issued via an event source 354 over an event
11 wire 356 to the event sink 326 of the primary SQL module 312. The “stop” event
12 directs the primary SQL module 312 to halt operation.

13 The fail-over policy module 310 issues an “assign owner” (AO) event from
14 an event source 358, over the event wire 360 to the event sink 342 of the store
15 316. The assign owner event directs the storage mechanisms to switch to allowing
16 access by the secondary SQL module 314, rather than the primary SQL module
17 312. The fail-over policy module 310 also issues a “start” event from event source
18 362 over event wire 364 to the event sink 334 of the secondary module 314. The
19 start event directs the secondary SQL module to start operation in place of the
20 primary SQL module.

21 The SQL database module 300 illustrates how the base model components
22 and exemplary model extensions—modules, ports, wires, stores, event sources,
23 event sinks, and event wires—may be arranged and interconnected to form a
24 complex module. The developer specifies the characteristics associated with each
25 component according to the prescribed schema. The complex module may in turn

1 be added to other simple or complex modules to form other complex modules.
2 Eventually, the largest complex module becomes the Internet Service, which may
3 then be used to form a blueprint for deploying to the data center.

4 Fig. 4 shows a simplified application 400 for an online retailer. The
5 application 400 includes a front end module 402, a catalog module 404, an order
6 processing module 406, and a fulfillment module 408. The application 400 also
7 includes a customer database 410 and the fault-tolerant SQL database module 300.
8 Notice that the SQL database module 300 is the same as that shown in Fig. 3 to
9 illustrate how complex modules can be nested into even greater complex modules
10 to form an application.

11 The front end module 402 handles requests from clients who wish to shop
12 with the online retailer. The front end module 402 has a port 420 that
13 accommodates communications with external clients using the TCP/IP protocol
14 over the Internet. The front end module 402 also has an order port 422 to define a
15 communication exchange with the order processing module 406 and a catalog port
16 424 for communication flow to the catalog module 404. The ports 422 and 424
17 may be configured according to any of a variety of types, which support any of a
18 number of protocols including SOAP, TCP, or UDP. An event sink 426 is also
19 provided to receive a "new product" message from the catalog module 404 when a
20 new product has been added to the catalog.

21 The catalog module 404 provides catalog information that may be served
22 by the front end to the requesting clients. The catalog module 404 has a front end
23 port 430 connected via a wire 432 to the catalog port 424 of the front end module
24 402. The front end port 430 has a type that matches the catalog port 424. The
25 catalog module 404 also has an event source 434 for communicating the "new

1 product” messages over wire 436 to the event sink 426 of the front end module
2 402.

3 A SQL port 438 interfaces the catalog module 404 with the SQL database
4 module 300. The SQL port 438 has a type that utilizes the TDS protocol for the
5 communication exchange with the external port 302 of the SQL database 300.

6 The order processing module 406 has a front end port 440 to define a
7 communication interface with the front end module 402 via a wire 442. The order
8 processing module 406 also has a fulfillment port 444 to facilitate communication
9 with the fulfillment module 408 over wire 446 and a database port 448 to facilitate
10 communication with the customer database 410 via wire 450.

11 An event source 452 is provided at the order processing module 406 to pass
12 “order complete” events to the fulfillment module 408 via wire 454. These events
13 inform the fulfillment module 408 that an order is complete and ready to be filled.
14 A second event source 456 passes “new account” events to the customer database
15 410 via wire 458 whenever a new customer orders a product.

16 The fulfillment module 408 has an order port 460 to provide access to the
17 wire 446 to the order processing module 406 and a database port 462 to interface
18 with the customer database 410. The fulfillment module 408 also has an event
19 sink 464 to receive the “order complete” events from the order processing module
20 406.

21 The customer database 410 has an order port 470 to provide access to wire
22 450 and a fulfillment port 472 to facilitate communication with the fulfillment
23 module 408 via wire 474. The customer database 410 further has an event sink
24 476 to receive the “new account” events from the order processing module 406.
25

1 The modeling approach illustrated in Figs 3 and 4 is tremendously
2 beneficial because it allows developers and IDC operators to view the entire
3 Internet Service in terms of functional pieces independent of deployment scale.
4 The online retailer Internet Service 400, for example, requires a front end unit, a
5 catalog unit, an order processing unit, and a fulfillment unit regardless of whether
6 the retailer is handling 100 hits a day or 10 million hits per day.

7 The scale-independent nature frees the developer to focus on his/her little
8 piece of the Service. For instance, a developer assigned the task of building the
9 front end module 402 need only be concerned with writing software code to
10 facilitate response/reply exchanges. Any communication to and from the module
11 is defined in terms of order-related data being passed to the order processing
12 module 406 via the order port 422 and product data being received from the
13 catalog module 404 via the catalog port 424. The developer defines the data flow
14 to and from the order port 422 and the catalog port 424 according to their
15 respective associated protocol types.

16 The Internet Service 400 can then be used to construct a computer system
17 that hosts the online retailer. Initially, the online retailer may not receive very
18 much traffic, especially if launched away from the Christmas season. So, perhaps
19 the front end module 402 deploys initially to only a few computers to handle the
20 light traffic from the Internet. But, suppose that over time the site becomes more
21 popular and the Christmas season is fast approaching. In this situation, the online
22 retailer may authorize the IDC operator to add many more computers for the front
23 end tasks. These computers are equipped with software and configured to accept
24 HTTP requests for product information and to serve web pages containing the
25

1 product information. The computers are added (or removed) as needed, without
2 altering the basic description of the Internet Service 400.

3 4 **Computer-Based Modeling System and Method**

5 Fig. 5 shows an exemplary computer system 500 that implements modeling
6 software used to design Internet Services. The modeling computer may be
7 implemented as one of the nodes in a Internet Service, or as a separate computer
8 not included as one of the nodes. The modeling computer has a processor 502,
9 volatile memory 504 (e.g., RAM), and non-volatile memory 506 (e.g., ROM,
10 Flash, hard disk, optical, RAID memory, etc.). The modeling computer 500 runs
11 an operating system 510 and modeling system 512.

12 For purposes of illustration, operating system 510 and modeling system 512
13 are illustrated as discrete blocks stored in the non-volatile memory 506, although it
14 is recognized that such programs and components reside at various times in
15 different storage components of the computer 500 and are executed by the
16 processor 502. Generally, these software components are stored in non-volatile
17 memory 506 and from there, are loaded at least partially into the volatile main
18 memory 504 for execution on the processor 502.

19 The modeling system 512 includes a user interface 514 (e.g., a graphical
20 UI) that presents the pictorial icons of the model components 516 (e.g., modules,
21 ports, sources, sinks, etc.), a component schema database 518, a logical-to-
22 physical converter 520, and an instance-tracking database 522. The modeling
23 system 512 allows a developer to design an Internet Service by defining modules,
24 ports, wires, and event message schemes. The user interface 514 presents symbols
25 of the components 516, such as the symbols shown in Figs 2-4, and permits the

1 developer to arrange and interconnect them. The UI 514 may even support
2 conventional UI techniques as drag-and-drop operations.

3 The symbols depicted on the screen represent an underlying schema 518
4 that is used to define the model. For instance, a block-like module symbol is
5 associated with the characteristics of the functionality that the module is to
6 represent in the Internet Service. Thus, the developer may define a database
7 module that has characteristics pertaining to the kind of database (e.g., relational),
8 the data structure (e.g., tables, relationships), software (e.g., SQL), software
9 version, and so forth. Accordingly, by drafting the model on the UI, the developer
10 is architecting the entire schema that will be used to design the scale-independent
11 Internet Service.

12 Once the Internet Service is created, the logical-to-physical converter 520
13 converts the Service to a physical blueprint that details the number of computers,
14 software components, physical ports, and so forth. The converter takes various
15 parameters—such as how many site visitors are expected, memory requirements,
16 bandwidth requirements, processing capabilities, and the like—and scales the
17 Internet Service according to the schema 518 created by the developer. The
18 converter 520 specifies the number of computers needed to implement each
19 module, the number of disks to accommodate the stores, and the types of
20 communications protocols among the modules and stores. The identity of every
21 component instance is recorded in an instance-tracking database 522. Instances in
22 the instance-tracking database 522 include those for modules, port, wires, and
23 instances of model extensions such as stores, event ports, and event wires.

24 In one embodiment, the developer writes management policy, which issues
25 commands on the schema 518 to create new instances of modules, port, and wires

1 to deploy the Internet Service. Developers may choose to write management
2 policy instead of using fully automatic logical-to-physical converter 520 when
3 they want finer control over the growth and management of the Internet Service.
4 The management code issues commands using the namespace defined by the
5 schema 518, but the commands operate on individual module, port, and wire
6 instances. The commands are still dispatched through the converter 520, which
7 allocates nodes to the management policy. Whether operating automatically or
8 driven by management policy code, the convert 520 records in the instance-
9 tracking database 522 the individual instances of modules, port, and wires.

10 In this manner, the modeling system changes the development effort from a
11 node-centric approach for architecting Internet Services to an application-centric
12 approach. Within conventional node-centric methodology, the focus was on the
13 computers and how they were laid out. The Internet Services was then loaded
14 onto the nodes in an ad hoc manner. With the new application-centric approach,
15 the focus is initially on the Internet Service itself. The physical nodes used to
16 implement the Internet Service are derived in terms of the Service schema once it
17 is specified. The instance-tracking database 522 gives both developers and
18 operators information about how many instances of each module are running at
19 any time and how the modules are connected using port instances and wires
20 instances within the Service schema.

21 Fig. 6 shows a method for modeling a scale-independent Internet Service.
22 The method 600 may be implemented, for example, by the modeling system 512
23 executing on the modeling computer 500. In such an implementation, the method
24 is implemented in software that, when executed on computer 500, performs the
25 operations illustrated as blocks in Fig. 6.

1 At block 602, the modeling system 512 allows the developer to define the
2 modules and extensions, such as stores, that form the functional elements of the
3 Internet Service. The UI 514 enables the developer to create modules and
4 extensions, such as stores, and to define their characteristics as prescribed by a
5 predetermined schema. This entry process begins to construct the logical building
6 blocks of the Service.

7 At block 604, the modeling system 512 enables the developer to define the
8 ports for the modules and module extensions, such as stores. The developer
9 selects the type of ports. The modeling system ensures compatibility of ports that
10 are connected to one another. At block 606, the developer also defines other
11 extensions, such as events that may be passed among modules and module
12 extensions, such as stores. For example, the developer creates event sources and
13 event sinks to accommodate the various events. At block 608, the developer uses
14 the modeling system 512 to interconnect the ports with wires and the port
15 extensions, such as event sources/sinks with wire extensions, such as event wires.
16 By joining the various modules and module extensions, such as stores, the
17 developer effectively forms a logical representation of the Internet Service.

18 At block 610, the modeling system 512 generates an Internet Service using
19 the graphical representation constructed by the developer through its associated
20 schema. The modeling system 512 generates the logical specifications associated
21 with the graphical model, including the characteristics of the modules and module
22 extensions, such as stores, as well as the types of the ports and port extensions,
23 such as event sources/sinks. The Internet Service provides a complete logical
24 representation of the Service that will eventually be implemented at the Internet
25

1 data center. The Internet Service description may be stored on disk or some other
2 form of computer-readable medium (block 612).

3 At block 614, the modeling system 512 converts the Internet Service
4 description to a physical blueprint that specifies the computers, the software run
5 by each of the computers, and the interconnections among the computers. This
6 physical blueprint may be used by the operator to install and manage the Internet
7 Service.

8 9 **Conclusion**

10 Although the description above uses language that is specific to structural
11 features and/or methodological acts, it is to be understood that the invention
12 defined in the appended claims is not limited to the specific features or acts
13 described. Rather, the specific features and acts are disclosed as exemplary forms
14 of implementing the invention.

CLAIMS

1. A method comprising:
representing hardware and software resources of a distributed computer system as model components; and
forming, from the model components, a logical scale-independent model of an application to be implemented by the distributed computer system.
2. A method as recited in claim 1, wherein each model component represents one or more similar resources.
3. A method as recited in claim 1, wherein each model component is depicted in a graphical user interface as a graphical icon.
4. A method as recited in claim 1, wherein the model components have an associated schema that specifies the hardware and software resources represented by the model components.
5. A method as recited in claim 1, wherein the model components comprise a module that is representative of a behavior of the application that is implemented using the hardware and software resources.
6. A method as recited in claim 1, wherein the model components comprise a store that is representative of persistent data storage.

1 7. A method as recited in claim 1, wherein the model components
2 comprise a port that is representative of a communication access point for the
3 model components.

4
5 8. A method as recited in claim 1, wherein the model components
6 comprise a wire that is representative of an allowable communication connection
7 between model components.

8
9 9. A method as recited in claim 1, wherein the model components are
10 selected from a group comprising:

11 a module that is representative of a behavior of the application that is
12 implemented using the hardware and software resources;

13 a port that is representative of a service access point for the module or the
14 store; and

15 a wire that is representative of an allowable communication connection
16 between two or more ports.

17
18 10. A method as recited in claim 9, wherein the group of the model
19 components further comprises at least one of:

20 a store that is representative of persistent data storage;

21 an event source that is representative of a logical connection point for the
22 module or the store from which event messages originate;

23 an event sink that is representative of a logical connection point for the
24 module or the store to receive the event messages; and
25

1 an event wire that is representative of an interconnection between the event
2 source and the event sink.

3
4 11. A method as recited in claim 1, further comprising storing the scale-
5 independent model on a computer-readable medium.

6
7 12. A method as recited in claim 1, further comprising converting the
8 scale-independent model into a blueprint of the server data center, the blueprint
9 specifying the hardware and software resources used to physically implement the
10 application.

11
12 13. A computer-readable medium storing computer-executable
13 instructions that, when executed on a computer, perform the method of claim 1.

14
15 14. A method comprising:
16 defining individual model components as abstract functional operations that
17 are physically implemented by one or more computers and one or more software
18 programs executing on the computers, the model components having an associated
19 schema dictating how the functional operations are specified;

20 interconnecting the model components to logically connect the functional
21 operations; and

22 generating a scale-independent application from the interconnected model
23 components and the associated schema.

1 **15.** A method as recited in claim 14, wherein each model component is
2 depicted in a graphical user interface as a graphical icon.
3

4 **16.** A method as recited in claim 14, wherein the defining comprises
5 entering, via a user interface, a description of elements needed to implement the
6 functional operations.
7

8 **17.** A method as recited in claim 14, wherein each graphical resource
9 represents a set of resources provided by the computers and the software
10 programs, the resources being scalable from one to many.
11

12 **18.** A method as recited in claim 14, wherein the model components are
13 selected from a group comprising:

14 a module that is representative of a behavior of the application;
15 a port that is representative of a communication access point for the
16 module; and
17 a wire that is representative of an interconnection between two or more
18 ports.
19

20 **19.** A method as recited in claim 18, wherein the group of the model
21 components further comprises:

22 a store that is representative of persistent data storage;
23 an event source that is representative of a logical connection point for the
24 module or the store from which event messages originate;
25

1 an event sink that is representative of a logical connection point for the
2 module or the store to receive the event messages; and

3 an event wire that is representative of an interconnection between the event
4 source and the event sink.

5
6 **20.** A method as recited in claim 14, further comprising storing the
7 application on a computer-readable medium.

8
9 **21.** A method as recited in claim 14, further comprising converting the
10 scale-independent application into a blueprint that specifies the computers and the
11 software programs used to physically implement the application.

12
13 **22.** A computer-readable medium storing computer-executable
14 instructions that, when executed on a computer, perform the method of claim 14.

15
16 **23.** A method comprising:
17 representing hardware and software resources of a distributed computer
18 system as model components; and
19 associating the model components with a schema dictating how the
20 hardware and software resources are specified.

21
22 **24.** A method as recited in claim 23, wherein the model components are
23 selected from a group comprising:

24 a module that is representative of a behavior that is implemented using the
25 hardware and software resources;

1 a port that is representative of a communication access point for the module
2 and the store; and

3 a wire that is representative of an interconnection between two or more
4 ports.

5
6 **25.** A method as recited in claim 24, wherein the group of the model
7 components further comprises:

8 a store that is representative of persistent data storage;

9 an event source that is representative of a logical connection point for the
10 module or the store from which event messages originate;

11 an event sink that is representative of a logical connection point for the
12 module or the store to receive the event messages; and

13 an event wire that is representative of an interconnection between the event
14 source and the event sink.

15
16 **26.** A method as recited in claim 23, further comprising creating a scale-
17 independent application from the model components and the associated schema.

18
19 **27.** A method as recited in claim 26, further comprising converting the
20 scale-independent application into a blueprint that specifies the hardware and
21 software resources used to physically implement the application on the distributed
22 computer system.

23

24

25

1 **28.** A modeling system, comprising:
2 a set of model components that represent hardware and software resources
3 of a distributed computer system;
4 a schema associated with the model components that dictate how the
5 resources are specified; and
6 a user interface to enable a developer to create an application by selecting
7 and interconnecting the model components and specifying the functionality of the
8 model components in accordance with the schema.

9
10 **29.** A modeling system as recited in claim 28, further comprising a
11 converter to convert the application to a blueprint that specifies the hardware and
12 software resources used to physically implement the application on the distributed
13 computer system.

14
15 **30.** A computer-readable medium comprising computer-executable
16 instructions that, when executed on one or more processors, direct a computing
17 device to:

18 represent hardware and software resources of a distributed computer system
19 as model components;

20 associate the model components with a schema dictating how the hardware
21 and software resources are specified; and

22 create an application by specifying the functionality of the model
23 components in accordance with the schema and interconnecting the model
24 components.
25

1 **31.** A computer-readable medium as recited in claim 30, further
2 comprising computer-executable instructions that, when executed on one or more
3 processors, direct a computing device to convert the application to a blueprint that
4 specifies the hardware and software resources used to physically implement the
5 application on the distributed computer system.

6
7 **32.** A system comprising:
8 means for representing hardware and software resources of a distributed
9 computer system;
10 means for specifying how the resources represented by the model
11 components are specified; and
12 means for selecting and interconnecting the model components and
13 specifying the functionality of the model components to create an application.

1 **ABSTRACT**

2 A modeling system permits developers of applications for distributed
3 computer system, such as those used in server data centers or Internet data centers
4 (IDCs), to architect their hardware and software in an abstract manner. The
5 modeling system defines a set of components that represent abstract functional
6 operations of the application that will eventually be physically implemented by
7 one or more computers and one or more software programs executing on the
8 computers. Associated with the model components is a schema that dictates how
9 the functional operations are to be specified. From the model components, the
10 developers can create logical, scale-independent models of the applications that
11 may be implemented by the distributed computer system. The application is scale-
12 independent in that the application is invariant in respect to the number of
13 computers and software programs that my eventually be used to implement it. The
14 application may subsequently be converted to a physical blueprint that specifies
15 the number and type of hardware/software resources and the physical layout of the
16 distributed computer system.

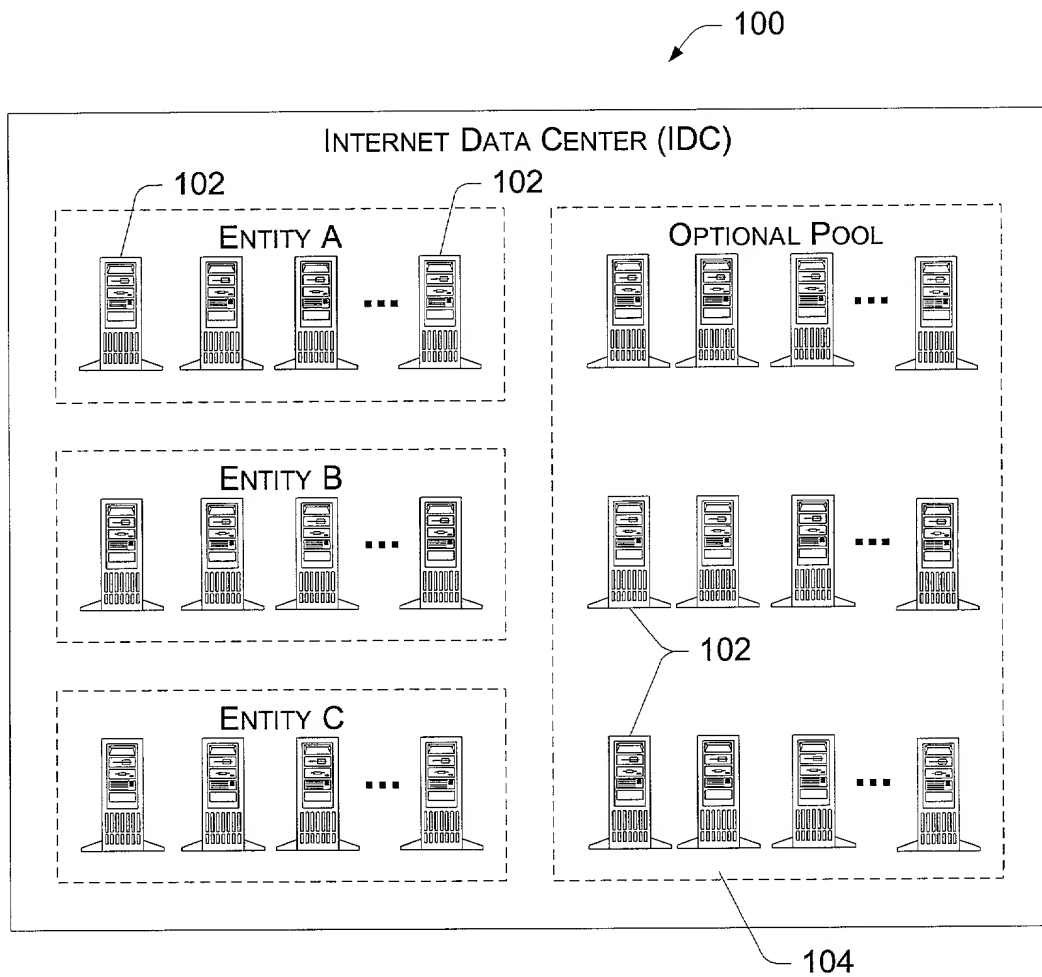


Fig. 1
Prior Art

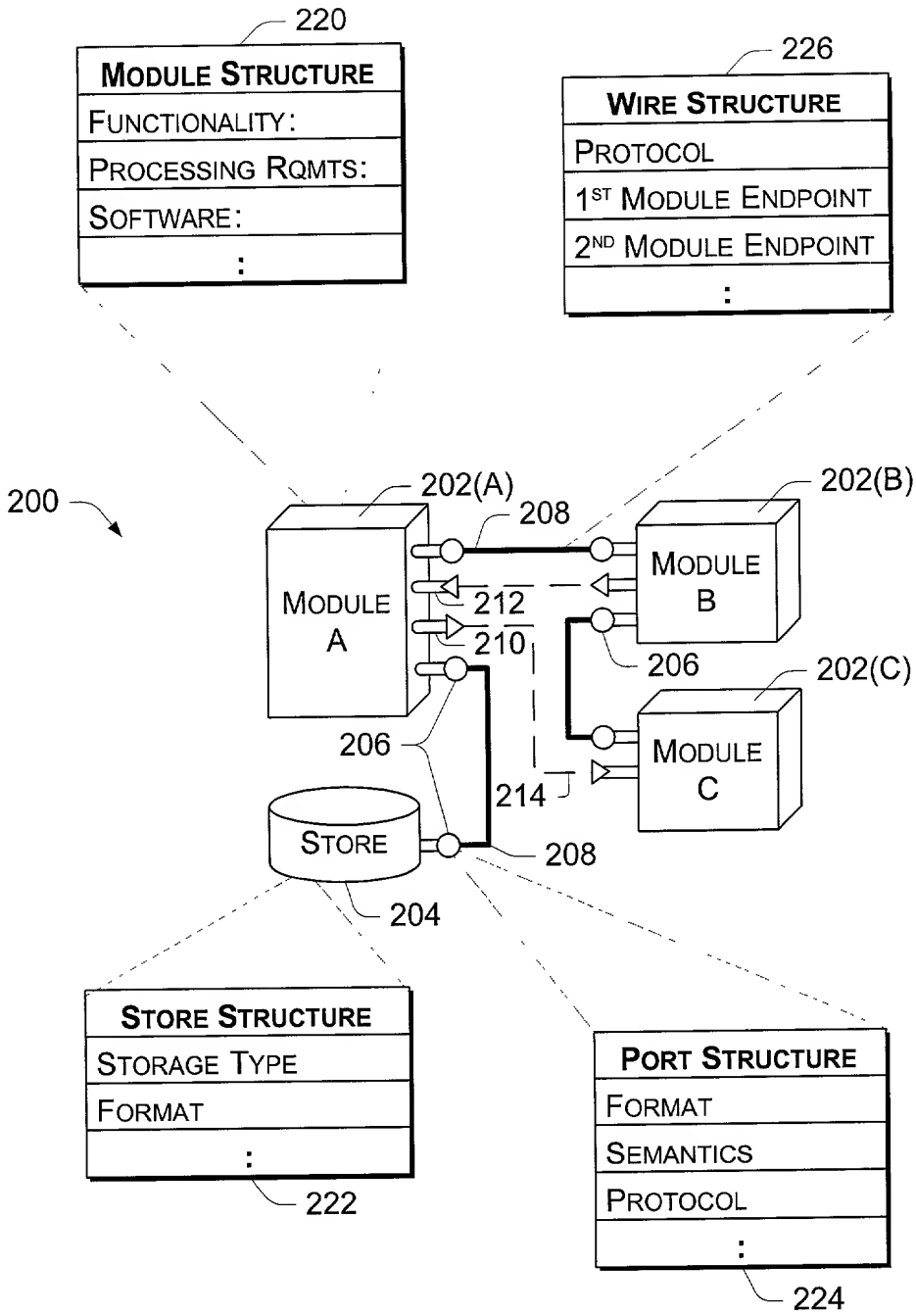


Fig. 2



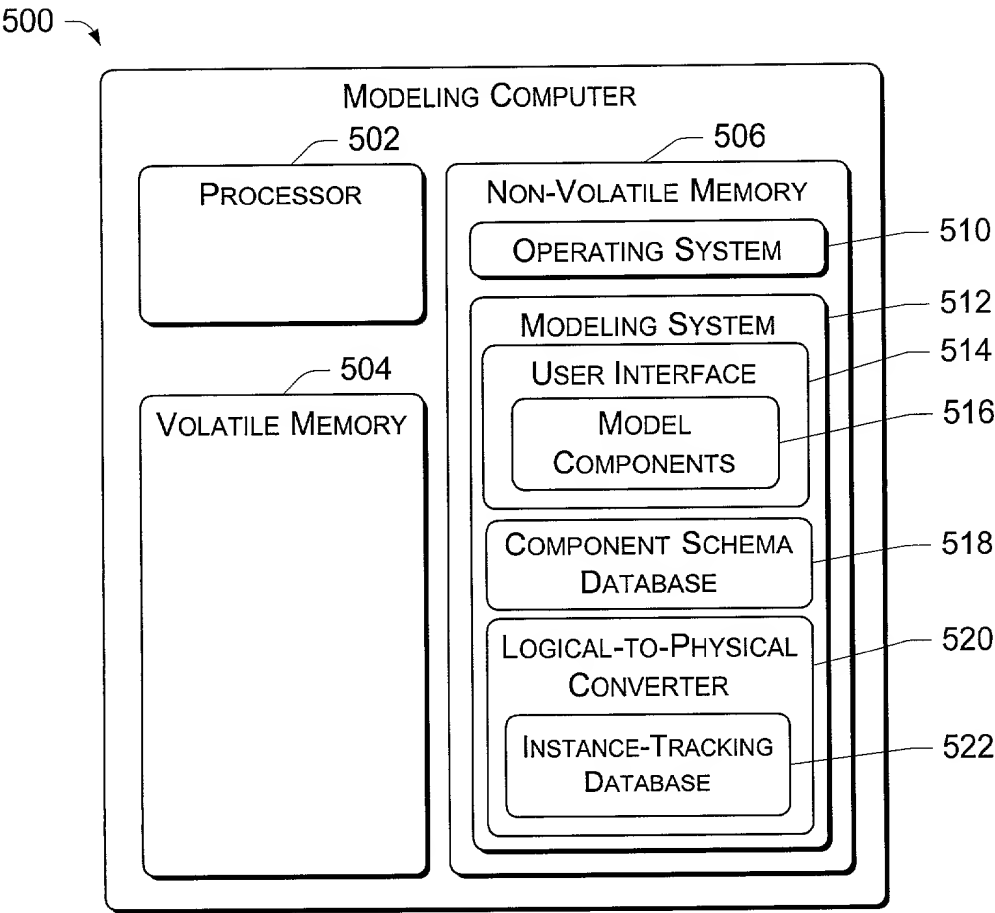
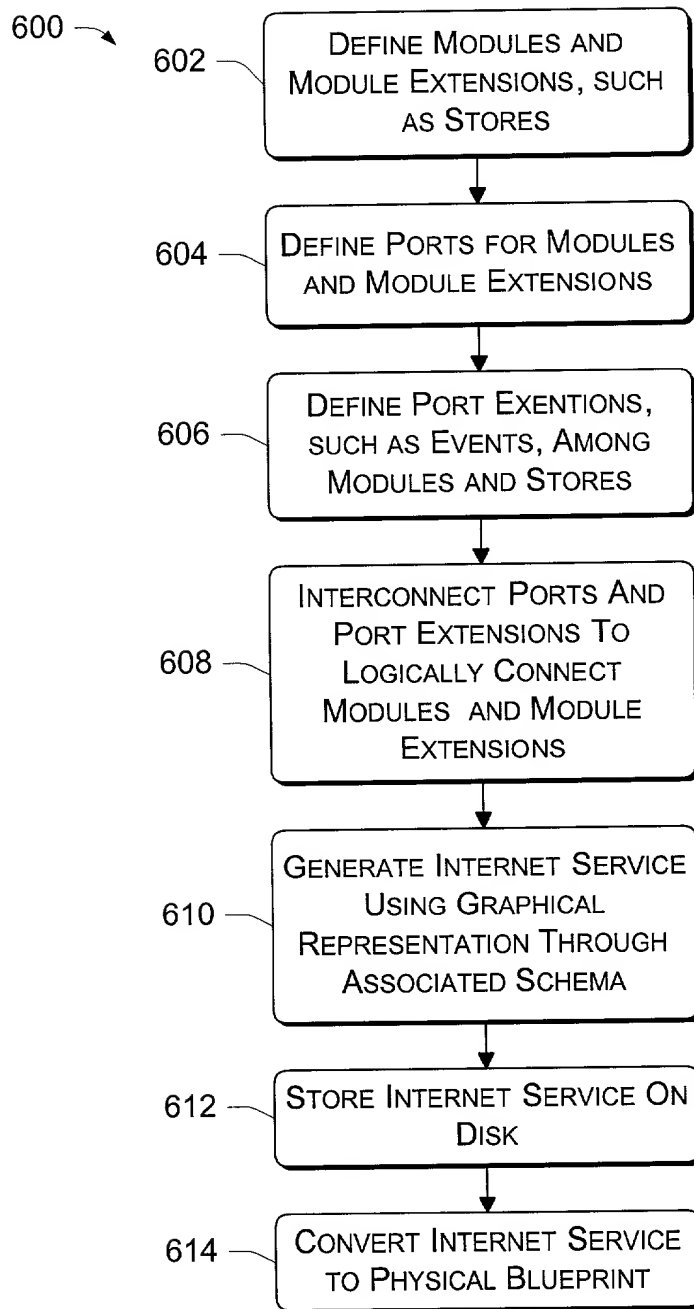


Fig. 5

*Fig. 6*